# DYNAMIC RIGHT-SIZING: TCP FLOW-CONTROL ADAPTATION

**Mike Fisk**, Los Alamos National Laboratory and University of California San Diego
mfisk@lanl.gov
**Wu-chun Feng**, Los Alamos National Laboratory
feng@lanl.gov

Los Alamos NATIONAL LABORATORY

## Abstract

**Network bandwidth has kept pace with the widespread arrival of bandwidth intensive applications such as streaming media and grid computing, but the TCP flow-control implementations in most operating systems make it difficult or impossible for applications to take advantage of high-bandwidth WANs. Dynamic Right-Sizing is an operating system technique for automatically tuning TCP to solve this problem. Compared to previous work, Dynamic Right-Sizing is more efficient and transparent and applies to a wider set of scenarios by simultaneously supporting network-bound senders, application-bound receivers, and both high- and low-bandwidth links.**

## Introduction

Grid and networking researchers continue the practice of manually optimizing TCP buffer sizes to keep the network pipe full [6, 3], and thus achieve acceptable performance over the wide-area network, whether for bulk-data transfer or in support of computational grids, data grids, or access grids. Not only is this process cumbersome, but the result of tuning window sizes for a particular pair of hosts is sub-par performance for connections with larger delay-bandwidth products and the misappropriation of scarce resources to connections with smaller delay-bandwidth products[4].

Consequently, we propose an operating system technique called Dynamic Right-Sizing that eliminates the need for this manual process. Compared to previous work on this problem, our solution is more efficient and both more transparent and widely-usable to applications.

In addition, Dynamic Right-Sizing enables conservative buffer management on senders and receivers. This addresses many of the TCP scalability problems with systems such as large web servers that handle thousands of simultaneous connections to a diverse set of peers.

In short, Dynamic Right-sizing lets the receiver estimate the sender's congestion window size and use that estimate to dynamically change the size of the receiver's window advertisements. As a result, the sender will be congestion-window-limited rather than flow-control-window-limited.

## Performance Tests

Figure 1 shows the relative speeds of Dynamic Right-Sizing, a manually tuned over-sized window, and a connection with the default window size. All measurements were made between network-bound Linux 2.4 systems with Gigabit Ethernet interfaces on a simulated WAN link with a 100ms round-trip delay.

Over-sizing is more aggressive in the beginning since the congestion window doubles with each acknowledgement, while Right-Sizing doubles the window once per round-trip. However, the over-size connection quickly incurs packet loss and reduces its rate to the same rate used by Right-Sizing. Thus Dynamic Right-Sizing achieves the same instantaneous throughput for most of the connection. Meanwhile, the default configuration is more than 13 times slower for this 78MB transfer.

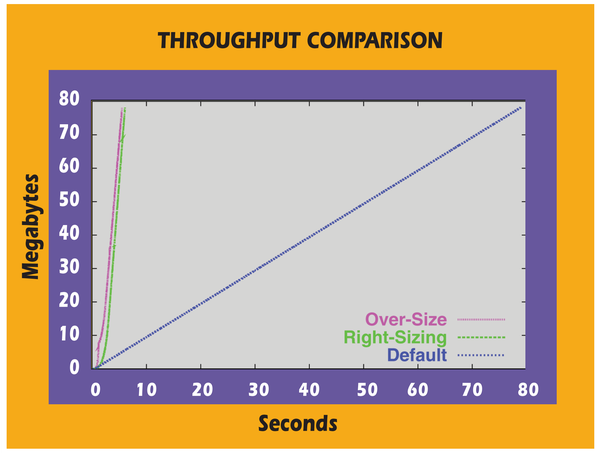


figure 1

Figure 2 shows how Dynamic Right-Sizing grows the receive window initially and then flattens off at twice the usable window size. The Flightsize is the amount of sent but unacknowledged data in transit.

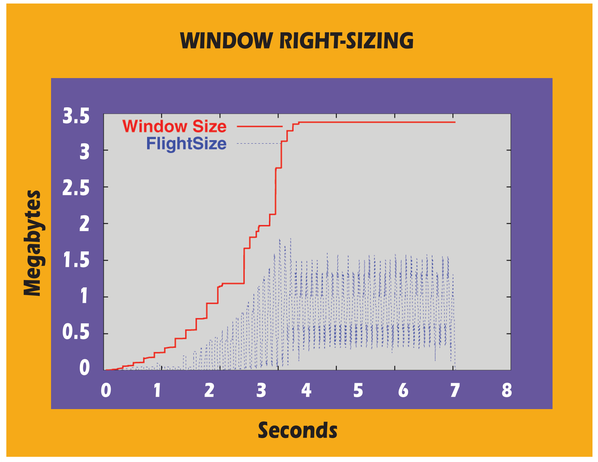


figure 2

## Need for Right-Sizing

| BOTTLENECK | SIZE OF BUFFERS & RECEIVE WINDOW | | |
|---|---|---|---|
| | Under-Size <demand | Right-Size =demand | Over-Size >>demand |
| Receiving App | unused bandwidth | no problems | over-allocated send buffers over-allocated receive buffers |
| Network | unused bandwidth | no problems | over-allocated send buffers |
| Sending App | unused bandwidth | no problems | no problems |

demand=min(Application-to-Application TCP-friendly throughput x delay, Congestion Window)

## Explanation:

**Unused bandwidth**:
The window advertised by the receiver is smaller than the bandwidth-delay product of the network. The sender will send a full window's worth of data but will then stall waiting for an acknowledgement from the receiver before it can send more. This leads to bursty stop-and-go transmissions punctuated by periods of unused bandwidth.

**Over-allocated send buffers:**
The sending application is generating data faster than TCP can transmit. The operating system's send buffers quickly fill, no matter how large they are. In order to send a full window's worth of data, the sender must use one window's worth of buffer. Some additional buffer allows the scheduling of the application to be decoupled from TCP, but using significantly more buffers space for this connection is unnecessary and may starve other connections.

**Over-allocated receive buffers:**
Receive buffers are used in two cases:
- To buffer acknowledged data until it can be delivered to the receiving application. This buffer allows the application scheduling to be decoupled from TCP, but only needs to be large enough to cover scheduling latency. If the data is arriving at a sustained rate that is faster than the receiving application given to the connection will be filled with no benefit and at the potential expense of the starvation of other connections.
- To buffer out-of-order data following packet loss. Assuming correct SACK operation, only two bandwidth-delay products worth of data is necessary to correct a small number of drop events.

## Comparison of Solutions

**Dynamic Right-Sizing (solves 5 of 5 problems)**
Dynamic Right-Sizing prevents both under-sized windows and oversized windows by bounding the window size by the congestion window and the measured throughput of the actual TCP connection. The kernel has better information about the buffer needs of individual connections, and can then fairly allocate buffers accordingly.

**Auto NCFTP (solves 3+ of 5 problems)**
AutoNCFTP[2] sets the send and receive buffers and receive window equal to the initial bandwidth-delay product of the network. This avoids problems with under-sized windows, but the bandwidth measurements will frequently be larger than the amount of bandwidth actually usable by TCP's congestion control mechanisms. As a result superfluous buffers space may be used on the sender and receiver for buffering data. This superfluous space may starve other connections, but not nearly to the degree of TCP Auto-Tuning.

**TCP Auto-Tuning (solves 3 of 5 problems)**
Auto-tuning[4] prevents under-sized windows by using over-sized windows. Fair-share algorithms are used to manage buffer space, but these allocation decisions are not based on the best window-size for a connection and the resulting amount of buffer space that is actually useful to each individual connection. As a result superfluous buffers space may be used on the sender and receiver for buffering data.

**Linux 2.4 (solves 2 of 5 problems)**
The Linux 2.4 kernel prevents over-using send buffers when the bottleneck is the receiver or network. This is done by starting connections with small send windows and growing them in proportion to the amount of data necessary to fill the network pipe, as affected by congestion control.

## Comparing Dynamic Right-Sizing and AutoNCFTP

Both Dynamic Right-Sizing and AutoNCFTP[2] adjust the receive buffer so that it is no longer a bottleneck to performance. However there are several important differences that stem from the fact that Dynamic Right-Sizing is a kernel modification to the TCP stack, while AutoNCFTP demonstrates user-space code that could be included in network applications running on unmodified operating systems.

**TCP-friendly Throughput vs. Available Bandwidth**
- The initial available bandwidth measured by AutoNCFTP is an upper bound to the TCP-friendly throughput, but is an overestimate of the amount of buffer used during TCP slow-start. Available bandwidth is also an over-estimate if the bottleneck in the connection is the sender or receiver rather than the network.
- Because AutoNCFTP is a user-space process, it has no standard way of determining its current congestion window.

**Passive vs. Active Measurement**
- AutoNCFTP requires 15KB of non TCP-friendly measurement traffic and as much as 2 seconds of setup latency. The Linux 2.4 solution was specifically targeted at short-lived connections such as typical web connections. In this context, the measurement overhead of AutoNCFTP is enormous.

**Kernel vs. User Space Deployment**
- AutoNCFTP can be deployed on systems even if it is not feasible to run a modified kernel, or to even obtain a modified kernel from the vendor. But systems that can run Dynamic Right-Sizing gain improved buffer management for all applications on a system without requiring any changes or recompilation of the applications.
- With AutoNCFTP, only modified applications provide information to the kernel about their buffer needs. This prevents the kernel from most effectively using memory when it is in shortage.

## Passive Measurement

**Round-Trip Time:**
To perform Dynamic Right-Sizing, it is necessary for the receiver to know the round-trip time. In a typical TCP implementation, the round-trip time is measured by observing the time between when data is sent and an acknowledgement is returned [1]. But during a bulk-data transfer, the receiver might not be sending any data and would therefore not have a good round trip time estimate. For instance, an FTP data connection transmits data entirely in one direction.

A system that is only transmitting acknowledgements can still estimate the round-trip time by observing the time between when a byte is first acknowledged and the receipt of data that is at least one window beyond the sequence number that was acknowledged. If the sender is being throttled by the network, this estimate will be valid. However, if the sending application did not have any data to send, the measured time could be much larger than the actual round-trip time. Thus this measurement acts only as an upper-bound on the round-trip time and should be be used only when it is the only source of round-trip time information.

**TCP-friendly Throughput:**
TCP-friendly throughput is measured by simply observing the progress of the connection for a round-trip time.

## Summary

- **Dynamic Right-Sizing has been implemented in Linux 2.2 and 2.4 kernels**
- **Measurement overhead is minimal: 24 bytes per connection**
- **Dramatic improvement in data transfer throughput, equivalent to manual tuning**
- **Improved buffer management and memory utilization for all network connections**
- **Our algorithms and kernel implementation provides several advantages over competing implementations**

## References

**[1]** Van Jacobson, "Congestion Avoidance and Control," in Proceedings of SIGCOMM '88, pp. 314– 329, Aug.1988

**[2]** Jian Lui and Jim Ferguson, "Automatic TCP socket buffer tuning," in SC 2000 Research Gems, Nov. 2000, http://dast.nlanr.net/Features/Autobuf/.

**[3]** Jamshid Mahdavi, "Enabling high performance data transfers on hosts," Webpage, http://www.psc.edu/networking/perf_tune.html.

**[4]** Jeff Semke, Jamshid Mahdavi, and Matt Mathis., "Automatic TCP buffer tuning," Proceedings of SIGCOMM '98, pp. 315–323, Oct. 1998.

**[5]** Brian L. Tierney, "TCP tuning guide for distributed application on wide area networks," ;login, vol. 26, no. 1, Feb. 2001.

**Visit us on the web at: http://public.lanl.gov/radiant/**

RADIANT

**Research and Development in Advanced Network Technology**